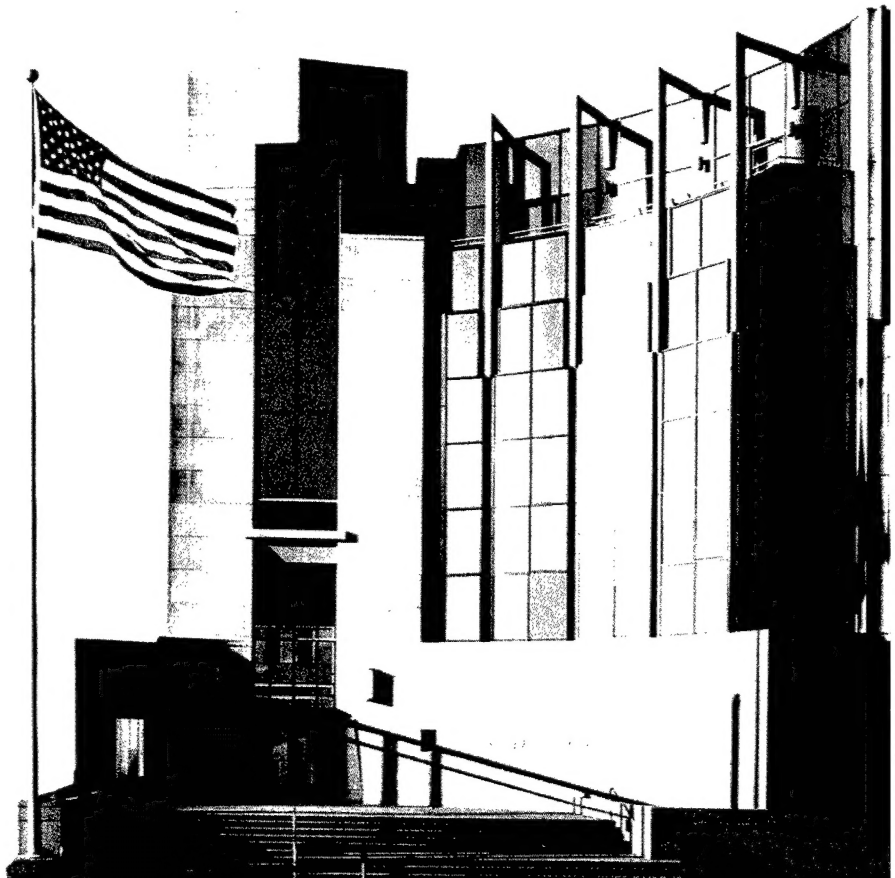
Carnegie Mellon
Software Engineering Institute

PECT Infrastructure: A Rough Sketch

Scott Hissam
James Ivers

December 2002

TECHNICAL NOTE
CMU/SEI-2002-TN-033



Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

PECT Infrastructure: A Rough Sketch

CMU/SEI-2002-TN-033

Scott Hissam
James Ivers

December 2002

Predictable Assembly from Certifiable Components Initiative

Table of Contents

List of Figures

List of Tables

Abstract

A prediction-enabled component technology (PECT) is an approach to achieving predictable assembly from certifiable components. A PECT consists of a component technology that has been extended with one or more reasoning frameworks that are used to predict how assemblies of components will behave. Developing and using a PECT involves a number of different activities, many of which are practical only when supported by automation. This paper investigates the nature of PECT infrastructures, summarizes the activities that a PECT infrastructure should support, and proposes a design for the tools that make up a PECT infrastructure. This paper also considers the reusability of such an infrastructure by evaluating the impact that three possible changes to a PECT have on its infrastructure.

1 Introduction

The Predictable Assembly from Certifiable Components (PACC) Initiative at the Software Engineering Institute (SEI[SM])[1] has been investigating a development activity for building systems from components where the runtime behavior of those systems (or assemblies of components) is predictable. An assembly is predictable (with respect to some property) if its behavior can be inferred from the properties of components and their patterns of interaction. A component is certifiable if the same component properties can be obtained or validated by independent third parties.

Our investigations into PACC have resulted in two specific applications of our approach that we call prediction-enabled component technology (PECT). These applications are documented in *Packaging Predictable Assembly with Prediction-Enabled Component Technology* [Hissam 01] and *Predictable Assembly of Substation Automation Systems: An Experiment Report* [Hissam 02]. A PECT extends the notion of component technology with one or more reasoning frameworks such that assemblies of components are guaranteed to be analyzable—and therefore predictable—with respect to those frameworks. More thorough discussions of PACC and PECT are available in *Volume I: Market Assessment of Component-Based Software Engineering* [Bass 01], *Volume II: Technical Concepts of Component-Based Software Engineering* [Bachmann 00], and *Volume III: A Technology for Predictable Assembly from Certifiable Components.*[2]

In our development of a PECT, we were without an underlying infrastructure to support it. We had to design and implement various tools to support various activities, including

- component specification: capturing component-level behaviors regarding some property (e.g., component execution time)
- component assembly: "wiring" together components to form assemblies
- component and assembly measurement: capturing and recording observations of component and assembly execution
- pre- and post-execution analysis: transforming constructed assemblies into forms that are analyzable, making predictions, and validating those predictions

1. SEI is a service mark of Carnegie Mellon University.

2. Wallnau, K. *Volume III: A Technology for Predictable Assembly from Certifiable Components* (CMU/SEI-2003-TR-009). Pittsburgh, PA: Software Engineering Institute, to be published.

Furthermore, the "glue" that would have allowed us to take artifacts and work products from each stage of PECT development had to be developed "just in time." This required constant translation of component and assembly information from one tool to another and rerunning experiments and predictions—effort that could have been reduced or eliminated with a supporting PECT infrastructure.

Based on this experience, we gained an appreciation of how essential it is to have an infrastructure to support a PECT and how much work developing such an infrastructure can be. This combination makes the cost of adopting a PECT significant if no infrastructure is already available. As part of our continuing research into PECTs, we will be developing multiple PECTs and would like to minimize the effort spent on infrastructure development. Moreover, our ultimate objective is to transition PECTs to the software community. With a constant eye toward our transition goal, we must simplify and reduce the effort needed to use PECTs whenever possible.

We believe that different PECTs share some common infrastructure needs. In this paper, we explore the activities that any PECT infrastructure must support, and we propose a design for such an infrastructure that promotes reusing portions of it. We evaluate the reusability potential in our design by proposing three scenarios that are representative of the significant challenges of reusing a PECT infrastructure, and we consider how each scenario affects the proposed infrastructure design.

1.1 About This Report

This document is our initial attempt to understand the requirements for a PECT infrastructure. We do this by looking at what we have accomplished in the past and what we want to accomplish in the future. Looking back, we take the experience from our past PECTs and distill the development activities that would have been facilitated by the existence of specialized tools; we also consider the roles and activities that were actually being carried out. Looking forward, we immediately recognize the need for an infrastructure that will support not only the development of PECTs, but also their usage. Additionally, we acknowledge that developing a new infrastructure for each PECT is impractical, and we begin to consider how to design an infrastructure such that sizeable portions are reusable across PECTs.

This document is a statement of what we believe a PECT infrastructure should do, what construction and analysis it should support, who it should support, and how it might be structured.

1.2 Structure of This Report

Section 2 summarizes the concepts embodied in a PECT. The roles and activities that a PECT infrastructure should support are found in Section 3. Section 4 illustrates an infrastructure

design in terms of specific tools and their interactions. This design is then evaluated in Section 5 against three change scenarios to consider each tool's reusability in the face of each change. Finally, Section 6 closes with a brief discussion of how we intend to approach the development of our next PECT infrastructure.

2 Summary of PECT Concepts

A PECT extends the notion of a component technology with one or more reasoning frameworks such that assemblies of components are predictable with respect to those frameworks. In Figure 1, we use a Unified Modeling Language (UML) class diagram to illustrate how a component technology relates to reasoning frameworks in a PECT.

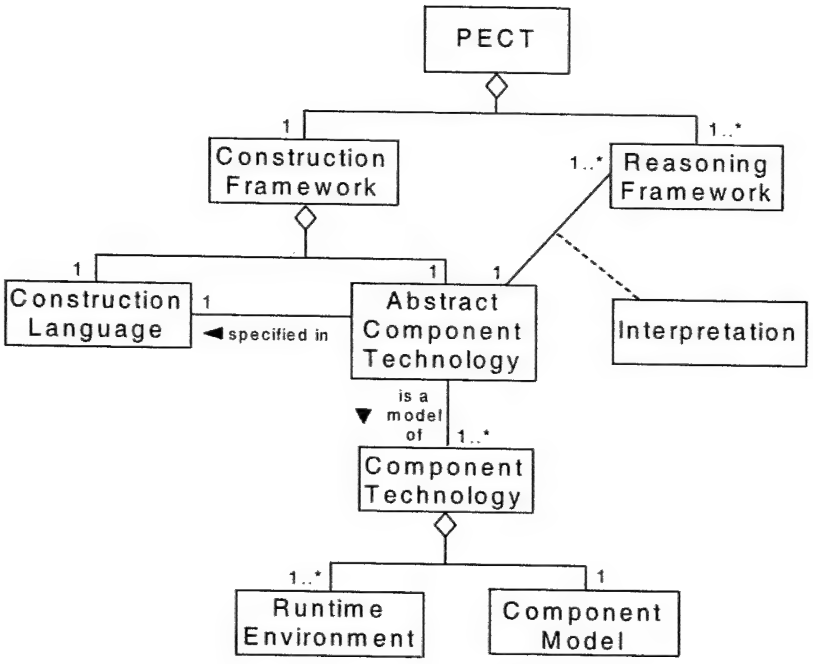


Figure 1: UML Class Diagram of PECT Concepts

A component technology consists of a component model and one or more runtime environments. The component model specifies allowable component types, interaction mechanisms, services provided by the runtime environment, and constraints among them all. A runtime environment is an execution environment that enforces aspects of the component model. A runtime environment plays a role analogous to that of an operating system, serving as the context in which components execute. Different runtime environments for the same component technology enforce the same component model, but may differ in terms of quality attributes, such as performance or reliability.

A component technology is incorporated into a PECT by means of a construction framework that consists of an abstract component technology (ACT) and a construction language. An ACT is a description of a particular component technology in a construction language. ACTs

are described using a common language—a construction language—to allow the same tools to be used with PECTs containing different component technologies.

The construction language is also used to describe assemblies constructed in accordance with the ACT and associated reasoning frameworks. A construction language includes the syntactic elements needed to capture three kinds of information:

1. the topology of an assembly (the composition of components that defines the assembly's structure)
2. the behavior of each component in the assembly, the interaction mechanisms defined by the component model, and the services provided by the component technology's runtime environments
3. arbitrary property descriptions, required by specific reasoning frameworks, that are attached to various syntactic elements, such as components or interactions

Each reasoning framework included in a PECT embodies the concepts and theories needed to analyze, and hence predict, certain emergent properties of an assembly of components. For each reasoning framework, an interpretation is defined that relates the concepts of the ACT to those of the framework. An interpretation is used during development to translate an assembly specification, as documented using the construction language, to a specification that can be used with the interpretation's reasoning framework.

3 Roles and Activities

Before we can describe the infrastructure that is needed to support a PECT, we need to understand the different roles PECT stakeholders play and how they use PECTs to do their jobs. We examine the differences among these roles by listing some of the activities stakeholders undertake. For each activity, we note infrastructure needs.

3.1 Roles

While much of the PACC work to date has focused on developing and validating a PECT, we recognize that these are not the end goals. We develop a PECT because it helps us develop systems that behave predictably. Consequently, we consider how a PECT is used, as well as how it is developed and validated. We use the distinction between developing and using a PECT as a starting point in considering different roles that PECT stakeholders might undertake.

A *PECT developer* focuses on developing the technology needed to predict the behavior of assemblies of components. A PECT developer does not necessarily focus on a particular system to be built, but instead figures out how to apply particular analysis models to a class of related systems that use the same component technology. A PECT developer must also validate (with some degree of confidence) that predictions apply to a system constructed using the PECT. Ideally, a PECT developer also provides infrastructure that can be used to construct and analyze component assemblies.

A PECT developer assumes one of several subroles: component technology specialist, analysis specialist, PECT designer, or PECT validator. A *component technology specialist* has a thorough understanding of the component technology on which the PECT is built, defines the ACT in the construction language, and handles any infrastructure issues relating to the component technology or its runtime environments. An *analysis specialist* has a thorough understanding of a reasoning framework and handles any infrastructure issues relating to that framework. *PECT designers* are responsible for integrating one or more reasoning frameworks with a component technology; they coordinate with component technology and analysis specialists to constrain the use of the component technology and to provide interpretations from the ACT to the reasoning frameworks. A PECT designer also coordinates the development of any infrastructure provided with the PECT. A *PECT validator* is responsible for confirming that predictions made using the PECT are "correct." This typically involves gathering data from system executions and comparing it to PECT predictions, and may require the development of infrastructure to support data collection.

A *PECT user* focuses on developing a particular system and uses the PECT to predict the behavior of the component assembly used to implement the system. A PECT user assumes that a PECT is "correct"—that is, the PECT's predictions can be trusted to some specified degree of confidence. A PECT user works within constraints imposed by the component technology and associated reasoning frameworks.

A PECT user assumes one of several subroles: component developer, component certifier, or component assembler. A *component developer* implements individual components and must conform to the constraints imposed by the PECT. A *component certifier* assesses whether or not (or perhaps how well) an implemented component matches information in its specification, such as execution time or behavioral models. A *component assembler* combines components to form assemblies that meet some need. A component assembler then uses the PECT's reasoning frameworks to predict emergent properties of the assembly, make needed changes to the assembly, and so on until the predicted properties meet the requirements.

3.2 Activities

In this section, we sketch some of the activities performed by stakeholders assuming the subroles of PECT developers and PECT users. These activities are not complete and should not be interpreted as the presentation of a "PECT method."[3] They are, however, the activities we currently believe are likely to apply to most PECT uses, regardless of the order in which they occur.

Because we made this list to gain a better understanding of what type of PECT infrastructure would be useful, each activity is accompanied by a brief note regarding infrastructure that would be useful in performing the activity.[4] Our convention is to follow each activity with a short list of notes regarding infrastructure support for the activity.

3. More detailed method workflows have been identified, but a complete method has not yet been detailed.

4. In some cases, particularly for PECT developer activities, the infrastructure support noted may be for infrastructure that is *produced* by individuals in the role, rather than *used* by them.

3.2.1 PECT Developers

Component Technology Specialist

Activity: Develop or identify a component technology and associated runtime environment.

> **Infrastructure support needs:** The infrastructure needed for this activity is outside our scope of concern. We are not pondering a suite of tools that helps component technology specialists produce new component technologies.

Activity: Constrain the component technology during co-refinement to improve analyzability.

> **Infrastructure support needs:** Constraints on the component technology suggest a need for a tool that evaluates whether a component (or assembly) satisfies the constraints. Furthermore, such constraints require formal definitions, which may be used as input to constraint-checking tools.

Activity: Formalize the component technology's construction model (e.g., the interaction semantics) in the PECT's construction language.

> **Infrastructure support needs:**
>
> - Tools are needed to parse and check component and assembly specifications written in the PECT's construction language. Checks to perform would include ensuring that a specification does not violate the constraints imposed by the component technology.
> - A tool is needed to compile assembly specifications into composite behavioral models suitable for automated analysis.

Analysis Specialist

Activity: Develop or identify a reasoning framework, and modify it as needed during co-refinement.

> **Infrastructure support needs:** Developing or identifying a reasoning framework is an exercise that does not require PECT-specific tools. Many reasoning frameworks have their own accompanying tools. For those that do not, we do not consider infrastructure that helps analysis specialists build such tools.

PECT Designer

Activity: Work with customer to identify PECT goals/requirements.

> **Infrastructure support needs:** There are no infrastructure needs for this activity.

Activity: Use co-refinement to constrain or generalize the ACT or reasoning framework.

Infrastructure support needs: Any adjustments that result in constraints to which assemblies must conform must be reflected in the infrastructure. A tool that can evaluate whether a component or assembly specification conforms to such constraints is also needed.

Activity: Develop or identify a construction language for the PECT (e.g., CL [Ivers 02]).

Infrastructure support needs: The infrastructure (including parsers and compilers) must be able to work with the construction language.

Activity: Define interpretations from the ACT to the reasoning framework.

Infrastructure support needs: This activity suggests a need for tools that implement the interpretations. Each such tool must be able to process composed assembly specifications, ensure that those specifications conform to the interpretation's constraints, and produce specifications in the language of the interpretation's target reasoning framework.

Activity: Determine which infrastructure to distribute with the PECT; that is, decide which infrastructure tools should be available to PECT users.

Infrastructure support needs: Whatever infrastructure is to be distributed with the PECT must (obviously) exist. Any tools that do not already exist must be developed and integrated (to the degree desired) with the rest of the PECT infrastructure. (See Section 3.2.2 for more information on which pieces of infrastructure are likely to be useful to PECT users.)

PECT Validator

Activity: Determine which assembly or assemblies will be used to validate the PECT.

Infrastructure support needs:

- In some cases, synthetic assemblies can be used to represent actual assemblies. In such cases, tools that can generate a number of synthetic assemblies are needed.
- In other cases, real assemblies may be used. In these cases, a means to construct such assemblies is required, implying that the PECT validator needs to perform those activities normally categorized as PECT user activities and can share the same infrastructure needs. (For more information, see Section 3.2.2.)

Activity: Determine which data must be collected to validate the reasoning framework's predictions and then collect it.

Infrastructure support needs: This activity suggests the need for a tool that can collect the data necessary for validation. Such data is gathered from an execution within a particular runtime environment.

Activity: Process data and compare the results to PECT predictions to determine the statistical accuracy of the latter.

Infrastructure support needs: This type of data analysis may be supported by spreadsheets or statistical packages, so there may be no need for custom infrastructure.

3.2.2 PECT Users

Component Developer

Activity: Develop a component specification for an implemented component. This can happen in one of three ways, each with different infrastructure needs:

1. The developer could write a component specification first, and then use it to implement the component.

   Infrastructure support needs:

   - In some cases, a code generator is needed to produce code skeletons based on the behavioral information found in the component specification.
   - Additional tools are necessary to check that successive changes to the code do not violate the original specification.

2. A developer could implement the component first, and then derive a component specification from it.

   Infrastructure support needs: A model extractor might be used to produce a component specification based on the component implementation.

3. A developer could write a specification, independently implement the component, and then show that the implementation satisfies the specification.

   Infrastructure support needs: A test harness may be necessary to demonstrate that the behavior expressed in the component specification is also exhibited by the component implementation.

Activity: Annotate a component specification with property information that is required by PECT reasoning frameworks.

Infrastructure support needs:

- This activity requires a tool that allows such annotations to be supplied.
- There may be a need for tools that help the developer capture property values. For example, if a performance reasoning framework requires execution time properties, infrastructure that allows the component developer to collect this information is necessary.

Component Certifier

Activity: Certify that a component specification (that includes property values) accurately represents the component implementation.

Infrastructure support needs: We do not sufficiently understand what infrastructure is needed here. Perhaps the tools required would be the same as those tools used to validate a PECT (such as a data collection tool that could be used in conjunction with a test suite derived from the component specification).

Component Assembler

Activity: Select and compose components to form an assembly.

Infrastructure support needs:

- An environment that facilitates component assembly is necessary. Such a tool would likely be a graphical user interface (GUI) in which components can be selected, annotated, edited, and assembled.
- This activity suggests the need for a repository of components to select from when forming an assembly.
- A tool that can evaluate whether an assembly specification conforms to the constraints of construction and reasoning frameworks is needed.

Activity: Use reasoning frameworks to determine whether an assembly meets its emergent property requirements.

Infrastructure support needs:

- To use a reasoning framework, an assembly specification must first be interpreted for the reasoning framework. A tool is needed to perform that interpretation. This tool should ensure that only assembly specifications satisfying the constraints of the reasoning framework can be analyzed.

- A tool that automates analysis should be provided for each reasoning framework.

Activity: Deploy an assembly to the runtime environment.

Infrastructure support needs: This activity should be fully automated. While what is required to deploy an assembly to a runtime environment likely varies with runtime environments, example steps include copying files to a specific location, building a configuration file, and registering components with the runtime environment.

4 PECT Infrastructure

As discussed in Section 3, a PECT tool infrastructure supports many of the activities of both PECT developers and users. Based on our experience in developing PECTs, we have identified a core set of infrastructure tools, shown in Figure 2, that supports those activities. We recognize that, as we get more experience as both PECT developers and users, the number of infrastructure tools (and possibly their scope) will change.

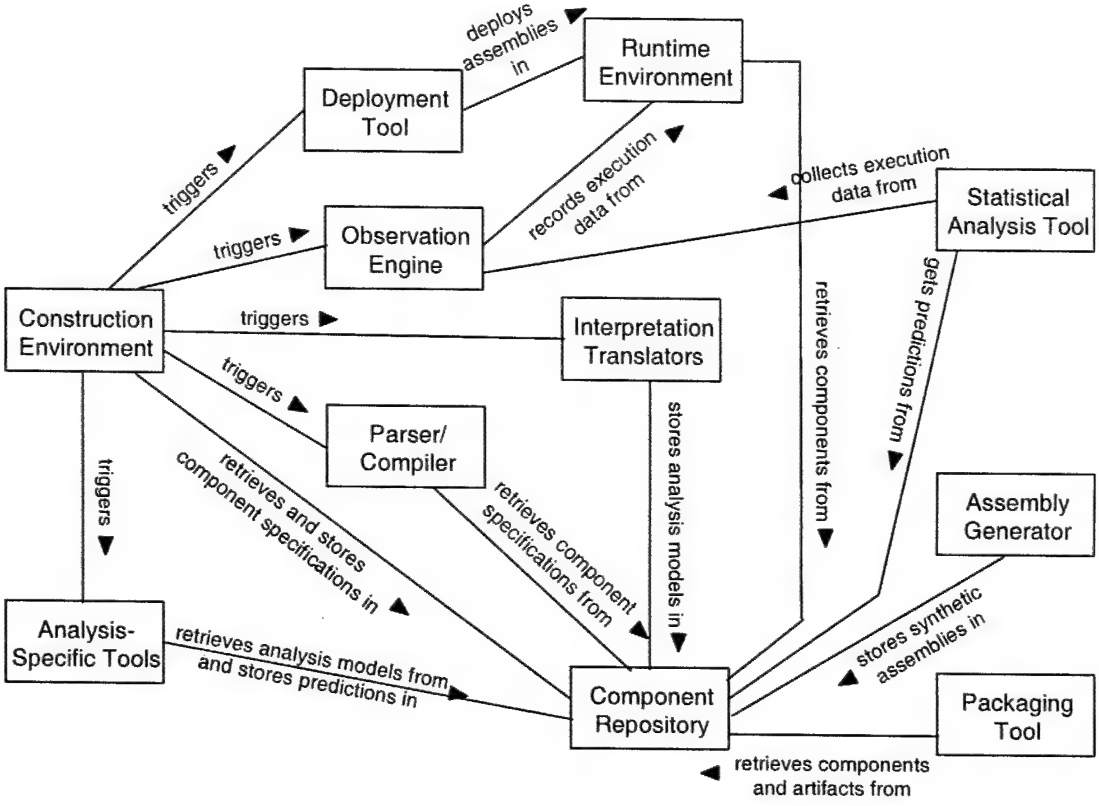


Figure 2: UML Class Diagram of PECT Infrastructure Tools

A PECT infrastructure is designed to support arbitrary empirical and formal reasoning frameworks (such as for average latency and safety) and arbitrary properties (such as latency, memory allocation, liveness, and reliability). Thus, the infrastructure shown in Figure 2 reflects the existence of analysis-specific tools for reasoning frameworks without identifying specific analysis tools. We envision that any such tool will be provided by the PECT developer. Hooks provided by the PECT infrastructure will then be used by PECT developers to integrate analy-

sis-specific tools. How such tools will interface with the remainder of the PECT infrastructure is uncertain.

Also uncertain is how all the tools within the PECT infrastructure interact. Figure 2 is one possibility. Of interest in this figure is the central role of the construction environment (described in Section 4.1). From our experience in *A Builder's Guide for Waterbeans Components*, the graphical builder was the focal point for assembly and execution, as this was where users selected components and composed assemblies [Plakosh 99]. From that original builder, hints were passed to the analysis tools so that predictions could be made and evaluated against observations gleaned from the runtime environment. In the PECT infrastructure we want to incorporate that working model. However, in some circumstances, we want to broaden the role of the graphical builder to coordinate other activities (such as deployment); in other circumstances we want to reallocate duties, such as interpretation and runtime, to other (now) externalized tools.

Figure 2 shows tools needed to satisfy the majority[5] of the developer and user needs identified in Section 3. PECT users, however, would typically only use a portion of these tools. Specifically, a PECT user uses all tools except for the statistical analysis tool, the assembly generator tool, and the packaging tool (at the right in Figure 2). These three tools are used primarily by PECT developers. Incidentally, those tools used by the PECT user are used much less by PECT developers. While PECT developers do need to "test out" the infrastructure that they build for PECT users, a PECT developer typically does not build an actual system.

Each tool from Figure 2 is briefly described in the remainder of this section.

4.1 Construction Environment

The construction environment is the visual integrated development environment (IDE) that supports component assembly via explicit interaction mechanisms. Essentially, the construction environment supports both PECT developers and users in the selection of components and their configurations as functioning assemblies.

Beyond its role as an end-user interface for the construction of assemblies, the construction environment can serve as a means to coordinate many PECT developer and user activities, such as

- assignment of property annotations to components as required by one or more reasoning frameworks upon which the PECT is based

5. Some infrastructure needs, such as the need for a tool that generates code from a component specification, are not included because they represent an area we are still exploring.

- constraint checking—that assemblies constructed are well formed (given the constraints of the ACT and associated reasoning frameworks) through integration with the parser
- generation of analysis models through integration with interpretation translators
- assembly execution (and debugging) through integration with the runtime environment
- packaging components and assemblies for deployment through integration with the deployment tool

4.2 Analysis-Specific Tools

Each reasoning framework is supported by one or more tools that automate the calculations needed to make predictions. We refer to each of these tools as an analysis-specific tool.

4.3 Runtime Environment

The runtime environment is the core of the execution environment for all components. Furthermore, the runtime environment conforms to the underlying component technology and constraints placed on the component technology so that PECTs are analyzable according to design and construction.

The runtime environment also supports collecting data about the execution of an assembly. In its simplest form, the runtime environment produces a trace of events showing the runtime activity of an assembly's execution. The runtime environment can associate additional information, such as a timestamp, with each event in the trace.

4.4 Deployment Tool

The deployment tool is used to prepare implemented components to run in the runtime environment. Deployment may include such activities as copying binaries to specific locations, providing the runtime environment with a description of how components are assembled, and setting configuration parameters for components or for the runtime environment itself.

4.5 Observation Engine

The observation engine records and processes assembly execution data collected by the runtime environment. The processing that can be performed depends on the raw execution data provided by the runtime environment; for example, if the runtime environment provides a trace of timestamped events, the observation engine can calculate the time elapsed between any two events in the assembly's execution. Given rich enough raw data, the observation engine could also calculate

- memory allocation
- processor utilization
- network utilization
- thread priority
- queue utilization
- throughput

Analysis tools constructed to use the services of the engine can observe when the assembly is actually executing within the runtime environment, or for post-run analysis, when the assembly has finished executing. The former is necessary for continuous or real-time monitoring of an assembly. The latter is necessary for historical analysis of prior assembly executions or for step-by-step analysis.

4.6 Parser/Compiler

The parser performs a series of well-formedness checks on component and assembly specifications written in the construction language. Checks include syntactic issues as well as whether the specification satisfies component technology constraints (e.g., that component inputs are not connected to the inputs of other components).

The compiler produces a composed specification for a well-formed assembly specification. Behavioral models for each component in the assembly are combined with behavioral models for the interaction semantics (provided by the component technology) to produce a composed specification for the assembly. This composed specification is used as the source for analysis model interpretations.

4.7 Interpretation Translators

An interpretation translator produces an analysis model from an assembly's construction language specification. However, a translator must first check that the assembly specification satisfies the assumptions of its reasoning framework. Such assumptions include the requirement that a specification include specific property annotations, or that components or the assembly satisfy constraints (e.g., a particular performance reasoning framework may assume that no asynchronous communication is used).

4.8 Component Repository

The component repository stores components and various artifacts associated with those components. The components stored in the repository are the actual, binary implementation of the components as loaded and executed by the runtime environment (see Section 4.3). The reposi-

tory also stores all component-associated artifacts needed to support the analysis of any component in the assembly. Such artifacts may include but are not limited to

- source code
- binary implementation
- component specification in the construction language
- analysis models (specifications in languages understood by analysis tools supporting reasoning frameworks)

Essentially, all tools within the infrastructure draw information from and record information to the repository.

4.9 Assembly Generator

The assembly generator is used during PECT validation to generate a number of synthetic assemblies that can be used to validate the accuracy of a reasoning framework's predictions. Each synthetic assembly is a collection of synthetic components annotated with property values required by a particular reasoning framework. The assembly generator varies the assignment of property values over different configurations of components to produce a variety of assemblies characteristic of those we want to analyze.

4.10 Statistical Analysis Tool

The statistical analysis tool is used during PECT validation to compare a series of reasoning framework predictions to observations of executing assemblies. The comparisons are used to estimate a measure of the confidence that a PECT user should have in the reasoning framework's predictions.

4.11 Packaging Tool

The packaging tool is used to produce a PECT distribution package that includes everything another user or group of users would need to use the PECT. The distribution package created by the tool contains all the components from the component repository as well as the artifacts necessary to support wholesale distribution of the PECT. The package may include

- component binary implementations
- runtime environment enforcing the PECT construction model constraints
- applicable component artifacts (specifications, properties, etc.)
- predictions

- sample or deployable assemblies
- scripts or post-deployment tools to perform active install-time property attribution (if necessary)
- test data
- other supporting analysis tools and configuration information

5 Infrastructure Change Scenarios

In this section, we propose three change scenarios that represent the significant challenges to reusing a PECT infrastructure. For each scenario, we evaluate how much each tool in the infrastructure would have to change to accommodate the scenario.

During co-refinement, multiple change scenarios may occur simultaneously. In particular, a new reasoning framework may be introduced that is more effective if changes are made to the component technology. This discussion, however, considers the impact of each change independent of any other changes.

5.1 New Reasoning Framework

In this scenario, a new reasoning framework is integrated into a PECT whose assumptions are already satisfied by the component technology.

Construction Environment

While the construction environment must integrate a new analysis tool(s) and an associated interpretation translator, the construction environment should be designed so that this integration can be accomplished using a simple plug-in mechanism. That is, no real changes are required to integrate new plug-ins.

Analysis-Specific Tools

A new analysis-specific tool must be provided for the new reasoning framework. No existing analysis tools for other reasoning frameworks in the PECT are affected.

Runtime Environment

Few or no runtime environment changes are needed to support a new reasoning framework. No execution semantics need to be changed, but the use of some component technology features may be restricted. It is possible that new execution data may need to be collected by the runtime environment to support validation of the PECT.

Deployment Tool

No changes are needed because the underlying component technology and runtime environment are not changed.

Observation Engine

An observation engine may be extended in response to the introduction of a new reasoning framework if new types of execution data must be processed to validate the predictions of the reasoning framework. If existing observations are sufficient to validate predictions, no changes are needed.

Parser/Compiler

No parser or compiler changes are needed to support a new reasoning framework because the construction language requires no changes.

Interpretation Translators

A new interpretation translator must be developed for the new reasoning framework. No existing interpretation translators for other reasoning frameworks in the PECT are affected.

Component Repository

No changes to the repository are needed; all repositories used with PECTs should already possess the ability to store arbitrary artifacts linked to a component or assembly.

Assembly Generator

A new reasoning framework requires the assembly generator to vary a different set of property values over the assemblies generated.

Statistical Analysis Tool

Although what is being predicted is different for a new reasoning framework, the process of comparing predictions to observations does not change. In other words, the raw data used by this tool changes, but the tool itself does not.

Packaging Tool

No changes are needed.

5.2 New Component Technology

In this scenario, a different component technology is used in a PECT, and existing reasoning frameworks are assumed to be compatible with the new component technology. While it is also possible to consider a scenario in which a component technology is simply tweaked, this scenario focuses on a large semantic change to a component technology.

Construction Environment

With the introduction of a new component technology, the construction environment must be integrated with a different set of tools (e.g., deployment tools). Any new interaction semantics introduced by the new component technology may require the introduction of new symbols (e.g., symbols for pins) to be used in the graphical editor. New interaction semantics may also require new topological constraints that may be enforced by the editor.

Analysis-Specific Tools

No changes to analysis tools are needed as long as the reasoning frameworks are compatible with the new component technology (i.e., the component technology does not violate any assumptions made by the analysis tools).

Runtime Environment

A new component technology requires a new runtime environment.

Deployment Tool

Because a new component technology requires a new runtime environment and because a new runtime environment may require different actions during deployment, a new deployment tool may be needed.

Observation Engine

The observation engine requires minimal changes to work with a new runtime environment. The data processing performed by the observation engine stays the same, but how it collects the data from the runtime environment may change.

Parser/Compiler

The parser and compiler must change to reflect the new component technology. Different constraints are enforced as well-formedness checks in the parser, and the compiler uses the component technology's formalized interaction semantics to produce composed models.

Interpretation Translators

Regardless of the specific component technology, an interpretation operates over a composed assembly model, which is expressed in terms of the construction language. Thus, no changes to the interpretation translators are necessary.

Component Repository

No changes to the repository are needed; all repositories used with PECTs should already possess the ability to store arbitrary artifacts linked to a component or assembly.

Assembly Generator

The assembly generator requires minor changes to generate only the assemblies that are valid under the constraints of the new component technology.

Statistical Analysis Tool

No changes to the statistical analysis tool should be needed, as the predictions and the observations stay the same when a new component technology is used.

Packaging Tool

No changes are needed.

5.3 New Construction Language

In this scenario, a different construction language is used in a PECT. Such a change could be largely syntactic, could represent a change in the underlying computational model of the formal language used to describe behavior, or could mean using a different architectural style or metaphor for describing components and assemblies. For the purpose of considering the impact on infrastructure tools, we assume that a change in construction language is significant (i.e., not merely syntactic), but that the new construction language is still based on the concepts of component assembly via explicit interaction mechanisms.

Construction Environment

Because the construction environment is the tool in which users document components using the construction language, at least moderate changes are required to support a different language. However, the portion of the construction environment that integrates other tools in the infrastructure remains unchanged.

Analysis-Specific Tools

No changes are needed for the analysis-specific tools. While the interpretation translators may be severely affected, the output of the translators remains the same. This output is what the analysis-specific tools operate on.

Runtime Environment

Because the runtime environment executes component implementations rather than specifications, no changes are needed.

Deployment Tool

No deployment tool changes are needed since the component implementation and the runtime environment remain unchanged.

Observation Engine

The observation engine may require minor changes if it uses information from component or assembly specifications. For example, if a user can select events from an assembly specification to observe during execution, a change to the construction language may require observation engine changes to enable the engine to understand the new language.

Parser/Compiler

A new parser and compiler are needed. The types of constraints to be checked and the work performed in compiling a composed model remain conceptually unchanged.

Interpretation Translators

All interpretation translators are affected by a change in construction language. If the construction language changes in a syntactic manner, only minor changes to the translators are needed. However, if the underlying formal language changes, translators are affected significantly.

Component Repository

No changes to the repository are needed; all repositories used with PECTs should already be able to store arbitrary artifacts linked to a component or assembly.

Assembly Generator

Minor (syntactic) changes would be required to enable the assembly generator to produce assemblies described in the new construction language.

Statistical Analysis Tool

No changes to the statistical analysis tool are needed because the predictions and the observations remain unchanged.

Packaging Tool

No changes are needed.

5.4 Summary of Scenario Impacts

Table 1 provides a summary of the effect that each change scenario has on each infrastructure tool.

Table 1: Summary of Change Scenario Impacts

Infrastructure Tool	New Reasoning Framework	New Component Technology	New Construction Language
Construction environment	N	M	M
Analysis-specific tools	H	N	N
Runtime environment	V	H	N
Deployment tool	N	H	N
Observation engine	V	L	L
Parser/compiler	N	M	H
Interpretation translators	H	N	H
Component repository	N	N	N
Assembly generator	L	L	L
Statistical analysis tool	N	N	N
Packaging tool	N	N	N
Key: H = high impact M = moderate impact L = low impact N = no impact V = variable impact			

6 Conclusion

In this paper, we have

- listed the activities that we believe any PECT infrastructure must support
- proposed a set of interacting tools forming such an infrastructure
- examined the reusability of tools in a PECT infrastructure in the face of three significant changes

This paper contains our current understanding of PECT infrastructures; we plan to use this information to guide the development of our next PECT infrastructure. As we continue to develop and research PECTs, our understanding may change.

In this paper, we examined change scenarios and the reusability of tools in a PECT infrastructure for both short-term and long-term reasons. In the short term, we will be producing several PECTs to further research and validate ideas. Because we would prefer to focus on the ideas rather than the tools, reusing as much of the infrastructure as possible leaves us with more time available for the "real work." In the long term, one factor that might influence the success of PECTs is the difficulty of developing a PECT for a particular set of needs. Infrastructure reuse may be able to lower infrastructure cost, but it is too early to know by how much.

When we develop infrastructure for our next PECT, we will use the following guidelines (sketchy though they are), any of which may result in infrastructure that is less capable than that described in this paper:

- Activities that are the most time-consuming or susceptible to errors will have priority over other activities.
- We will implement only those tools in the infrastructure for which no "workable" alternatives already exist.
- We may implement "bare-bones" tools for some pieces of the infrastructure to save time (e.g., we may use a file system as a simple repository).
- We will accommodate the types of changes we have hypothesized only when the effort involved in doing so is less than the cost of prototyping an alternative.

Once we have developed a few more PECTs and have more experience with how different PECTs impact infrastructure needs, we can revisit the issue of general PECT infrastructure.

One promising avenue to explore is the connection between PECTs and software product lines, and the applicability of proven reuse approaches from that movement.

References

URLs are valid as of the publication date of this document.

[Bachmann 00] Bachmann, F.; Bass, L.; Buhman, C.; Comella-Dorda, S.; Long, F.; Robert, J.; Seacord, R.; & Wallnau, K. *Volume II: Technical Concepts of Component-Based Software Engineering* (CMU/SEI-2000-TR-008, ADA379930). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr008.html>.

[Bass 01] Bass, L.; Buhman, C.; Comella-Dorda, S.; Long, F.; Robert, J.; Seacord, R.; & Wallnau, K. *Volume I: Market Assessment of Component-Based Software Engineering* (CMU/SEI-2001-TN-007, ADA395250). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn007.html>.

[Hissam 01] Hissam, S.; Moreno, G.; Stafford, J.; & Wallnau, K. *Packaging Predictable Assembly with Prediction-Enabled Component Technology* (CMU/SEI-2001-TR-024, ADA399793). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tr024.html>.

[Hissam 02] Hissam, S.; Hudak, J.; Ivers, J.; Klein, M.; Larsson, M.; Moreno, G.; Northrop, L.; Plakosh, D.; Stafford, J.; Wallnau, K.; & Wood, W. *Predictable Assembly of Substation Automation Systems: An Experiment Report* (CMU/SEI-2002-TR-031). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tr031.html>.

[Ivers 02] Ivers, J.; Sinha, N.; & Wallnau, K. *A Basis for Composition Language CL* (CMU/SEI-2002-TN-026, ADA407797). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tn026.html>.

[Plakosh 99] Plakosh, D.; Smith, D.; & Wallnau, K. *Builder's Guide for Water-Beans Components* (CMU/SEI-99-TR-024, ADA373154). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. <http://www.sei.cmu.edu/publications/documents/99.reports/99tr024/99tr024abstract.html>.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2002	Final

4. TITLE AND SUBTITLE

PECT Infrastructure: A Rough Sketch

5. FUNDING NUMBERS

F19628-00-C-0003

6. AUTHOR(S)

Scott Hissam, James Ivers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

CMU/SEI-2002-TN-033

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12.a DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited, DTIC, NTIS

12.b DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

A prediction-enabled component technology (PECT) is an approach to achieving predictable assembly from certifiable components. A PECT consists of a component technology that has been extended with one or more reasoning frameworks that are used to predict how assemblies of components will behave. Developing and using a PECT involves a number of different activities, many of which are practical only when supported by automation. This paper investigates the nature of PECT infrastructures, summarizes the activities that a PECT infrastructure should support, and proposes a design for the tools that make up a PECT infrastructure. This paper also considers the reusability of such an infrastructure by evaluating the impact that three possible changes to a PECT have on its infrastructure.

14. SUBJECT TERMS

prediction-enabled component technology, PECT, predictable assembly, predictable assembly from certifiable components, PACC, component technology

15. NUMBER OF PAGES

42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL